Bab 7

Query (II): Query Multitabel (JOIN, Subselect, UNION/ INTERSECT)

Seperti telah dijelaskan di Bab 2 mengenai model relasional, kekuatan utama database relasional adalah menyimpan data dalam tabel-tabel terpisah. Metode ini menghindarkan duplikasi data dan menghilangkan inkonsistensi serta menghemat ruang disk. Merelasikan antartabel menggunakan foreign key juga memiliki manfaat menjaga integritas dan kualitas kebersihan data. Pada contoh latihan di Bab sebelumnya, database *karyawan* "betulan" umumnya tidak terdiri hanya dari 1 tabel karyawan saja, melainkan bisa saja terpisah-pisah menjadi tabel *divisi* untuk mencatat daftar divisi yang ada (atau tabel *strukturorg* untuk mencatat struktur organisasi). Dengan demikian kolom *divisi* tabel *karyawan* sendiri diganti dengan kolom *id_divisi* yang merujuk ke tabel *divisi*. Manfaatnya? Database bisa menjaga agar user tidak memasukkan sembarang nama divisi melainkan hanya divisi yang terdaftar di tabel *divisi*. Demikian juga kolom-kolom lain seperti jabatan atau level bisa dipisahkan ke tabel lain. Dan seiring bertambah kompleks/ lengkapnya database, akan ada semakin banyak tabel (mis: tabel *gaji* untuk mencatat sejarah gaji dan tanggal naik gaji, tabel *orang* untuk memisahkan karyawan dan orang agar kalau terjadi mutasi atau kenaikan jabatan bisa ditrack dengan lebih mudah, dan lain sebagainya). Sebuah query akan dituntut tidak hanya mengambil data dari satu tabel saja melainkan dari beberapa tabel sekaligus.

Ada beberapa cara untuk melakukan query multitabel di SQL, yaitu menggunakan JOIN, subselect, dan operasi ala set (UNION, INTERSECT, dan lain sebagainya). Bab ini akan menjelaskan semua cara query multitabel tersebut.

7.1 Kombinasi Tabel (Produk Kartesius)

Cara pertama yang termudah untuk mengambil data dari beberapa tabel sekaligus adalah dengan sintaks produk kartesius.

```
— Resep 7-1-1: Produk kartesius beberapa tabel
SELECT ... FROM tabel1, tabel2[, ...];
```

Database akan menghasilkan jumlah baris sebanyak (jumlah baris *tabel1*) x jumlah baris *tabel2*) x .... Contoh:

```
CREATE TABLE t1 (i INT);
INSERT INTO t1 VALUES (1);
INSERT INTO t1 VALUES (2);
INSERT INTO t1 VALUES (3);
CREATE TABLE t2 (i INT);
INSERT INTO t2 VALUES (4);
INSERT INTO t2 VALUES (5);
mysql> SELECT * FROM t1,t2;
+------+------+
| i    | i    |
+------+------+
|    1 |    4 |
|    2 |    4 |
|    3 |    4 |
|    1 |    5 |
|    2 |    5 |
|    3 |    5 |
+------+------+
6 rows in set (2.03 sec)
```

`SELECT * FROM t1,t2` akan memberikan semua kolom *t1*, ditambah semua kolom *t2*, dan menghasilkan semua kombinasi baris-baris *t1* dan *t2*. Jika Anda ingin menyertakan kolom-kolom tertentu saja, bisa menggunakan sintaks `namatabel.namakolom` seandainya terdapat nama kolom yang sama (contoh: *t1.i* dan *t2.i*). Anda bisa

mengkombinasikan lebih dari 2 tabel. Anda juga bisa mengkombinasikan sebuah tabel dengan tabel itu sendiri (dalam kasus ini, Anda perlu memberi alias tabel pada salah satu tabel agar namanya jadi terbedakan). Beberapa contoh lain:

```
-- membuat tabel perkalian dari 1x1 s.d. 10x10
CREATE TABLE angka (i INT);
INSERT INTO angka VALUES
(1),(2),(3),(4),(5),(6),(7),(8),(9),(10);
mysql> SELECT t1.i AS angka1, t2.i AS angka2, t1.i*t2.i AS kali
       FROM angka t1, angka t2;
+--------+--------+------+
| angka1 | angka2 | kali |
+--------+--------+------+
|      1 |      1 |    1 |
|      2 |      1 |    2 |
|      3 |      1 |    3 |
...
|      8 |     10 |   80 |
|      9 |     10 |   90 |
|     10 |     10 |  100 |
+--------+--------+------+
100 rows in set (0.02 sec)

-- mendaftar kombinasi 3 warna yang berbeda, dengan syarat satu
harus
--   warna teduh, sisanya warna kontras/terang. tidak diinginkan
warna
--   hitam atau putih.
CREATE TABLE warna (hex CHAR(6), nama VARCHAR(32), sifat
VARCHAR(16));
INSERT INTO warna VALUES ('ff0000', 'merah', 'kontras');
INSERT INTO warna VALUES ('00ff00', 'hijau', 'kontras');
INSERT INTO warna VALUES ('0000ff', 'biru', 'kontras');
INSERT INTO warna VALUES ('ffff00', 'kuning', 'kontras');
INSERT INTO warna VALUES ('000000', 'hitam', 'kontras');
INSERT INTO warna VALUES ('ffffff', 'putih', 'kontras');
INSERT INTO warna VALUES ('ffcccc', 'pink', 'teduh');
INSERT INTO warna VALUES ('ccffcc', 'hijau muda', 'teduh');
INSERT INTO warna VALUES ('ffffcc', 'kuning muda', 'teduh');
INSERT INTO warna VALUES ('f0f0f0', 'abu-abu muda', 'teduh');
mysql> SELECT
```

```
        w1.nama warna1, w2.nama warna2, w3.nama warna3,
        CONCAT_WS('+', w1.hex, w2.hex, w3.hex) kode
      FROM warna w1, warna w2, warna w3
      WHERE
        – bukan hitam maupun putih...
        ('hitam' NOT IN (w1.nama, w2.nama, w3.nama)) AND
        ('putih' NOT IN (w1.nama, w2.nama, w3.nama)) AND
        – dua warna harus teduh, sisanya kontras
        (w1.sifat='kontras' AND w2.sifat='teduh'
         AND w3.sifat='teduh') AND
        – warna harus berbeda
        (w1.hex <> w2.hex AND w1.hex <> w3.hex AND w2.hex <>
w3.hex);
+--------+-------------+-------------+--------------------+
| warna1 | warna2      | warna3      | kode               |
+--------+-------------+-------------+--------------------+
| merah  | hijau muda  | pink        | ff0000+ccffcc+ffcccc |
| hijau  | hijau muda  | pink        | 00ff00+ccffcc+ffcccc |
| biru   | hijau muda  | pink        | 0000ff+ccffcc+ffcccc |
...
| hijau  | kuning muda | abu-abu muda | 00ff00+ffffcc+f0f0f0 |
| biru   | kuning muda | abu-abu muda | 0000ff+ffffcc+f0f0f0 |
| kuning | kuning muda | abu-abu muda | ffff00+ffffcc+f0f0f0 |
+--------+-------------+-------------+--------------------+
48 rows in set (0.00 sec)
```

7.2 JOIN

JOIN pada dasarnya sama dengan mengkombinasikan lebih dari satu tabel menggunakan cara sebelumnya, perbedaannya hanyalah: 1) perbedaan sintaks, tidak menggunakan sintaks `SELECT ... FROM tabel1, tabel2, ...` melainkan menggunakan sintaks `SELECT ... FROM tabel1 JOIN tabel2 ...`; 2) ada beberapa jenis dan variasi sintaks JOIN; 3) rata-rata JOIN memiliki klausa kondisi, yang pada dasarnya fungsinya sama dengan klausa WHERE namun dilakukan sebelum WHERE dan tujuannya untuk membantu database mengurangi jumlah kombinasi baris yang harus dilakukan.

Salah satu hal yang membuat pemula SQL bingung adalah mengenai JOIN ini. Ada beberapa jenis JOIN dan ditambah lagi ada beberapa jenis istilah yang *berbeda* untuk tipe JOIN yang *sama*. Pemula kadang kesulitan mengerti perbedaan tipe JOIN yang satu dengan yang lainnya.

Tabel 7-2-1 di bawah ini mendaftarkan tipe-tipe JOIN yang ada.

Nama	Nama lain	Memiliki klausa kondisi?	Catatan
CROSS JOIN		Tidak	Sama dengan produk kartesius. Sama dengan INNER JOIN ON (TRUE).
INNER JOIN, join dalam	JOIN	Ya	Hanya menghasilkan baris kombinasi jika kondisi terpenuhi.
LEFT OUTER JOIN	LEFT JOIN	Ya	Baris dari tabel di kiri selalu disertakan, tak peduli kondisi terpenuhi atau tidak.
RIGHT OUTER JOIN	RIGHT JOIN	Ya	Baris dari tabel di kanan selalu disertakan, tak peduli kondisi berhasil atau tidak. Sama dengan LEFT OUTER JOIN dengan membalik tabel kanan dan kiri.
FULL OUTER JOIN	FULL JOIN	Ya	Baris dari tabel kiri maupun kanan selalu disertakan, tak peduli kondisi berhasil atau tidak.
NATURAL INNER JOIN	NATURAL JOIN	Tidak	Sama dengan versi tanpa NATURAL, tapi versi NATURAL mencari kondisi otomatis berdasarkan nama-nama kolom.
NATURAL LEFT OUTER JOIN	NATURAL LEFT JOIN		
NATURAL RIGHT OUTER JOIN	NATURAL RIGHT JOIN		
NATURAL FULL OUTER JOIN	NATURAL FULL JOIN		
self join	Hanya istilah untuk JOIN yang menggunakan tabel yang sama di sisi kanan dan sisi kiri.		
straight join	Hanya istilah di MySQL yang menginstruksikan agar jangan mengutak-atik urutan JOIN (lihat manual MySQL).		

Melihat tabel di atas, sebagian dari Anda mungkin bertambah bingung. Namun sebetulnya hanya ada empat tipe JOIN yang utama yaitu INNER JOIN, LEFT [OUTER] JOIN, RIGHT [OUTER] JOIN, dan FULL [OUTER] JOIN; sisanya hanya variasi atau istilah saja.

Memahami JOIN

Sintaks JOIN adalah sbb.:

```
SELECT ... FROM tabel1 JOIN tabel2 [kondisi_join]
                    [JOIN tabel3 [kondisi_join]]
                    [...];
```

tabel1 disebut tabel "kiri" (karena ada di kiri kata kunci JOIN) sementara *tabel2* disebut tabel "kanan". Kedua tabel bisa saja tabel yang sama, di mana dalam kasus ini orang kadang menyebutnya self join (join dengan "diri sendiri"). JOIN dapat dirantaikan, sama seperti produk kartesian yang dapat mengkombinasikan tiga atau lebih tabel sekaligus.

kondisi_join dapat berbentuk:

```
ON ekspresi
```

atau

```
USING (nama_kolom1[, nama_kolom2[, ...]])
```

di mana sintaks USING di atas adalah shortcut bagi:

```
ON      tabel_kiri.nama_kolom1 = tabel_kanan.nama_kolom1
   [AND tabel_kiri.nama_kolom2 = tabel_kanan.nama_kolom2
   [AND ...]]
```

Sementara NATURAL JOIN adalah shortcut untuk USING di mana nama-nama kolom diambil dari kolom-kolom di tabel kiri dan kanan yang nama dan tipenya sama. Sehingga jika kebetulan nama-nama kolom yang ada sudah pas, kita dapat menyebutkan NATURAL JOIN dan tidak perlu lagi menambahkan klausa USING atau ON.

Kata kunci JOIN dapat diganti dengan berbagai tipe JOIN yang lain seperti yang telah disebutkan di kolom pertama di tabel 7-2-1. Dan seperti yang telah disebutkan sebelumnya, hanya ada empat tipe utama JOIN: INNER JOIN

(selanjutnya akan disebut JOIN saja), LEFT OUTER JOIN (selanjutnya disebut LEFT JOIN), RIGHT OUTER JOIN (selanjutnya disebut RIGHT JOIN), dan FULL OUTER JOIN (selanjutnya disebut FULL JOIN). Perbedaan di antara keempatnya adalah dalam kelakuan kala kondisi JOIN tidak terpenuhi. Mari melihat perbedaan keempat tipe ini dengan contoh:

```
DROP TABLE IF EXISTS t1;
-- t1 berisi angka dari 1 hingga 4
CREATE TABLE t1 (i INT);
INSERT INTO t1 VALUES (1),(2),(3),(4);

DROP TABLE IF EXISTS t2;
-- t2 berisi angka dari 3 hingga 6
CREATE TABLE t2 (i INT);
INSERT INTO t2 VALUES (3),(4),(5),(6);

-- 1. INNER JOIN
mysql> SELECT t1.i i1, t2.i i2 FROM t1 JOIN t2 ON t1.i=t2.i;
+------+------+
| i1   | i2   |
+------+------+
|    3 |    3 |
|    4 |    4 |
+------+------+
2 rows in set (0.00 sec)

-- 2. LEFT [OUTER] JOIN
mysql> SELECT t1.i i1, t2.i i2 FROM t1 LEFT JOIN t2 ON t1.i=t2.i;
+------+------+
| i1   | i2   |
+------+------+
|    1 | NULL |
|    2 | NULL |
|    3 |    3 |
|    4 |    4 |
+------+------+
4 rows in set (0.00 sec)

-- 3. RIGHT [OUTER] JOIN
mysql> SELECT t1.i i1, t2.i i2 FROM t1 RIGHT JOIN t2 ON t1.i=t2.i;
+------+------+
```

```
| i1   | i2   |
+------+------+
|    3 |    3 |
|    4 |    4 |
| NULL |    5 |
| NULL |    6 |
+------+------+
4 rows in set (0.00 sec)

-- 4. FULL [OUTER] JOIN
-- Catatan: FULL JOIN belum didukung MySQL, jadi hasil di bawah
hanya
--   hasil fiktif
mysql> SELECT t1.i, t2.i FROM t1 FULL JOIN t2 ON t1.i=t2.i;
+------+------+
| i1   | i2   |
+------+------+
|    1 | NULL |
|    2 | NULL |
|    3 |    3 |
|    4 |    4 |
| NULL |    5 |
| NULL |    6 |
+------+------+
6 rows in set (0.00 sec)
```

Bisa dilihat pada contoh di atas bahwa perbedaan di antara keempat tipe JOIN ini adalah: INNER JOIN mewajibkan kondisi terpenuhi dan jika tidak maka tidak akan menghasilkan baris. LEFT JOIN akan selalu menghasilkan baris-baris dari tabel kiri tak peduli apa pun dan jika di tabel kanan tidak ada baris sepadan maka nilainya akan digantikan dengan NULL; RIGHT JOIN sebaliknya akan berpihak pada tabel kanan dan jika di kiri tidak ada kecocokan maka akan diisikan NULL; FULL JOIN menggabungkan kelakuan LEFT dan RIGHT. FULL JOIN saat ini belum didukung MySQL, rencananya akan didukung di versi 5.1. RIGHT JOIN sebetulnya hanyalah sinonim untuk LEFT JOIN jika urutan tabel kiri dan kanan dibalik dan disediakan oleh bahasa SQL untuk kenyamanan saja (kadang di sebuah query lebih enak menyebutkan sebuah tabel di sisi kiri dan kadang lebih enak di sisi kanan). Bukti bahwa keduanya sama:

```
-- hasilnya sama dengan SELECT ... FROM t1 RIGHT JOIN t2 (#3 pada
--   contoh sebelumnya)
```

```
mysql> SELECT t1.i i1, t2.i i2 FROM t2 LEFT JOIN t1 ON t1.i=t2.i;
+------+------+
| i1   | i2   |
+------+------+
|    3 |    3 |
|    4 |    4 |
| NULL |    5 |
| NULL |    6 |
+------+------+
4 rows in set (0.82 sec)
```

Memilih Tipe JOIN

JOIN yang paling sering digunakan adalah JOIN biasa (alias INNER JOIN) dan LEFT JOIN (atau RIGHT JOIN, bergantung selera). FULL JOIN umumnya tidak begitu sering dipakai sehari-hari (ini sebabnya MySQL menangguhkan implementasi FULL JOIN-nya hingga versi 5.1). Produk kartesius atau CROSS JOIN juga lebih jarang dipakai, karena pada umumnya kita menginginkan join dengan kondisi tertentu, bukan kombinasi buta.

INNER JOIN seringkali dipakai jika kita ingin mengambil sebuah atribut yang terdapat pada tabel lain dan sekalian ingin memfilter hanya menginginkan baris yang memiliki atribut tersebut. Misalnya, kita ingin melakukan men-join tabel *karyawan* dengan tabel *email* dan sekalian hanya ingin memilih karyawan yang memiliki alamat email (memiliki baris di tabel *email*). Contoh lain, untuk memilih hanya anggota yang terdaftar dalam sebuah milis, kita dapat melakukan INNER JOIN antara tabel *milis* (berisi id dan nama tiap milis) dan tabel *anggota_milis* (yang berisi kolom *id_milis* dan *alamat_email*). Milis yang tidak memiliki anggota sama sekali tidak akan memiliki baris di tabel *anggota_milis* dan tidak akan disertakan oleh INNER JOIN.

LEFT JOIN umumnya dipakai untuk "mengambil atribut dari tabel lain *jika ada*." Jika di tabel sisi kanan tidak terdapat baris sama sekali, maka atribut yang dikembalikan akan bernilai NULL namun baris di kiri tetap ada. Misalnya kita ingin mendaftar nama-nama karyawan dari tabel karyawan beserta daftar bonus yang pernah diterima dan tanggalnya, *jika ada*. Daftar bonus ada di tabel *bonus* yang memiliki kolom *id_karyawan*, *tanggal*, *besar_bonus*. Jika seorang karyawan tidak pernah menerima bonus maka di tabel *bonus* tidak akan ada barisnya. Namun dengan LEFT JOIN kita akan tetap menampilkan si karyawan tersebut.

Mari melihatnya dengan contoh. Tabel *karyawan* adalah tabel yang kita buat sewaktu latihan di Bab 6 lalu.

```
CREATE TABLE bonus (
  id_karyawan INT NOT NULL
    – FOREIGN KEY (id_karyawan) REFERENCES karyawan(id)
    ,
  tanggal DATE NOT NULL,
  besar NUMERIC(18,4) NOT NULL,
  UNIQUE (id_karyawan, tanggal)
);
INSERT INTO bonus VALUES (1, '2003-12-31', 25000000);
INSERT INTO bonus VALUES (1, '2002-12-31', 20000000);
INSERT INTO bonus VALUES (4, '2003-12-31', 7500000);
INSERT INTO bonus VALUES (53, '2003-12-31', 7500000);

-- tampilkan nama dan jabatan karyawan yang gajinya 10jt ke atas
--   beserta bonus yang diterimanya tahun 2003 jika ada
mysql> SELECT
  CONCAT(nama_depan, ' ', nama_belakang) AS nama,
  jabatan,
  bonus.besar AS besar_bonus,
  bonus.tanggal AS tgl_bonus
FROM karyawan
LEFT JOIN bonus ON karyawan.id = bonus.id_karyawan AND
                   YEAR(bonus.tanggal) = 2003
WHERE gaji >= 10*1000000;
+--------------+------------------+-------------+-----------+
|nama          |jabatan           | besar_bonus | tgl_bonus |
+--------------+------------------+-------------+-----------+
|Koko Wacana   |Direktur Utama    | 25000000.0000 | 2003-12-31 |
|Nina Darman   |Wakil Direktur Utama| NULL        | NULL      |
|Wawan Setiawan |Manajer Marketing |  7500000.0000 | 2003-12-31 |
|Ema Hotimah   |Manajer Keuangan  |  7500000.0000 | 2003-12-31 |
+--------------+------------------+-------------+-----------+
4 rows in set (0.01 sec)
```

Bisa dilihat dari contoh di atas bahwa Nina Darman tidak menerima bonus (tidak ada baris untuk Nina Darman di tabel *bonus*). Namun karena kita menggunakan LEFT JOIN maka baris Nina tetap masuk karena masih memenuhi klausa WHERE. Jika kita menggunakan JOIN saja (INNER JOIN) maka hasilnya hanya 3 baris. Pada contoh di atas, kita harus memasukkan kriteria `gaji >= 10*1000000` di klausa WHERE, bukan JOIN karena kita tidak ingin memasukkan yang gajinya di bawah 10 juta (ingat sifat LEFT JOIN, jika kondisi join tidak terpenuhi maka baris tabel

kiri akan tetap diikutkan). Jika kita menggunakan INNER JOIN, maka kita boleh memasukkan kriteria `gaji >= 10*1000000` di klausa JOIN atau klausa WHERE dan hasilnya akan sama.

7.3 Subselect

Subselect (kadang disebut juga subquery atau nested select atau nested query) adalah query SELECT yang ada di dalam perintah SQL lain (entah itu perintah SELECT, atau UPDATE, atau INSERT, dan lain sebagainya).

Subselect baru didukung di MySQL 4.1, jadi semua query yang mengandung subquery yang ada pada contoh di Bab ini otomatis hanya berlaku untuk versi 4.1 ke atas.

Ada empat macam jenis subselect. Pertama, subselect scalar, yakni perintah SELECT yang hanya menghasilkan satu buah nilai skalar (satu buah "sel"). Misalnya adalah: `SELECT nama_depan FROM karyawan WHERE id=5` di mana perintah ini hanya berisikan satu kolom dan dijamin hanya akan mengembalikan 1 baris karena kolom *id* merupakan PK. Sehingga hasilnya hanyalah satu buah nilai. Dalam teori, subselect scalar dapat digunakan di mana saja nilai skalar dapat digunakan. Misalnya, di dalam WHERE:

```
-- cari karyawan yang pernah diberi bonus 2x atau lebih
mysql> SELECT nama_depan, nama_belakang FROM karyawan
WHERE (SELECT COUNT(*) FROM bonus WHERE id_karyawan=karyawan.id)
> 1;
+------------+--------------+
| nama_depan | nama_belakang |
+------------+--------------+
| Koko       | Wacana       |
+------------+--------------+
1 row in set (0.00 sec)
```

Di sini kita bisa melihat bahwa subquery bertindak seperti fungsi. Bayangkan fungsi `berapa_kali_bonus(id_karyawan)` yang akan mengembalikan nilai 0, 1, 2, dan lain sebagainya.

Subselect scalar juga dapat dipakai di ekspresi SELECT. Contohnya, variasi dari bentuk di atas:

```
mysql> SELECT
         nama_depan, nama_belakang,
```

```
        (SELECT COUNT(*) FROM bonus WHERE
id_karyawan=karyawan.id)
          AS berapa_kali_bonus
      FROM karyawan
      HAVING berapa_kali_bonus > 1;
```

Dalam kenyataannya, ada tempat-tempat tertentu di mana MySQL melarang kita menaruh subselect scalar, misalnya di klausa LIMIT.

Jenis subselect kedua adalah **subselect kolumnar**, yaitu perintah SELECT yang menghasilkan sebuah kolom. Contohnya adalah: `SELECT nama_depan FROM karyawan`. Perintah ini akan menghasilkan sebuah kolom berisi daftar nama depan karyawan, yang bisa dipandang juga merupakan sebuah list atau array. Subselect kolumnar dapat ditaruh di tempat yang mengharapkan list, misalnya dalam IN atau dalam ekspresi ANY dan ALL (dua yang terakhir ini belum didukung MySQL). Contoh:

```
-- Cari apakah ada pelanggan yang nama belakangnya sama dengan
--   salah satu karyawan
SELECT nama_depan, nama_belakang FROM pelanggan
  WHERE nama_belakang IN
    (SELECT DISTINCT nama_belakang FROM karyawan);
```

Jika Anda seorang programer, Anda bisa membayangkan subselect kolumnar ini seperti sebuah array dan klausa `WHERE nama_belakang IN (SELECT ...)` bisa dianalogikan dengan fungsi untuk mencari apakah sebuah nilai terdapat dalam array.

Jenis subselect ketiga adalah **subselect baris**. Sesuai namanya, subselect ini menghasilkan satu baris tunggal. Misalnya: `SELECT id, nama_depan, nama_belakang, gaji FROM karyawan WHERE id=1`, di mana hasilnya adalah sebuah baris yang terdiri dari tiga kolom. Saat ini jenis subselect ini jarang digunakan, tapi pada dasarnya subselect baris merupakan versi 1 baris dari subselect tabular (lihat jenis keempat di bawah).

Jenis terakhir adalah **subselect tabular** yang menghasilkan tabel atau relasi. Contohnya: `SELECT id, nama_depan, nama_belakang, gaji`. Subselect jenis ini dapat dipakai di tempat-tempat yang mengharapkan nama tabel, seperti di klausa FROM maupun di klausa JOIN. Contoh, kita bisa mengganti `LEFT JOIN bonus` pada contoh sebelumnya:

```
-- tampilkan nama dan jabatan karyawan yang gajinya 10jt ke atas
```

```
--   beserta bonus yang diterimanya tahun 2003 jika ada
mysql> SELECT
  CONCAT(nama_depan, ' ', nama_belakang) AS nama,
  jabatan,
  bonus.besar AS besar_bonus,
  bonus.tanggal AS tgl_bonus
FROM karyawan
LEFT JOIN bonus ON karyawan.id = bonus.id_karyawan AND
                   YEAR(bonus.tanggal) = 2003
WHERE gaji >= 10*1000000;
```

dengan:

```
... LEFT JOIN (SELECT id_karyawan, tanggal, besar FROM bonus) ON ...
```

Tentu saja contoh ini tidak terlalu berguna, karena si subselect hanya mengambil keseluruhan tabel dan tidak ada bedanya dengan isi si tabel itu sendiri. Tapi untuk query yang kompleks, kita dapat menghasilkan sekumpulan baris-baris yang telah difilter dan disortir dalam sebuah subselect tabular, untuk selanjutnya kita filter dan sortir lagi, atau dijoin dengan tabel lain, dan seterusnya. Kemungkinannya menjadi amat banyak dan fleksibel.

Subselect vs JOIN

Sebetulnya kalau dilihat, subselect lebih mudah dimengerti bagi seorang programer, karena mirip seperti fungsi atau loop. Sementara JOIN menuntut kita untuk memvisualisasikan mana tabel yang harus ditaruh di kiri/kanan dan jenis JOIN seperti apa yang kita inginkan. Namun dari kacamata si database sendiri, sebuah query SELECT yang mengandung subselect tabular pada dasarnya sama dengan JOIN atau kombinasi tabel. Bahkan dalam mengeksekusi sebuah query yang mengandung subselect, optimizer database pada umumnya memang mengkonversi dulu subselect tersebut menjadi JOIN agar lebih mudah dioptimasi. Namun kita tidak perlu mengetahui detail di balik layar ini.

Memilih menggunakan subselect atau JOIN bergantung pada selera Anda. Pada umumnya query yang mengandung subselect dapat ditulis dalam versi JOIN-nya, dan sebaliknya. Tapi kadang tidak semuanya bisa dikonversi.

7.4 UNION dan INTERSECT (dan Except)

Selain mengkombinasikan tabel (yang bersifat "perkalian"), kita juga dapat "menjumlahkan" atau "mengurangi" atau "mengiris" satu hasil query dengan query

lain, mirip dengan operasi himpunan. Perlu diingat bahwa memang sebuah relasi adalah set (tepatnya multiset atau bag).

```
-- Resep 7-4-1: Sintaks UNION
 SELECT ...
 UNION [ALL | DISTINCT] SELECT ...
[UNION [ALL | DISTINCT] SELECT ...]
[...]

-- Resep 7-4-2: Sintaks INTERSECT
--   Catatan: belum didukung oleh MySQL
SELECT ...
 INTERSECT [ALL] SELECT ...
[INTERSECT [ALL] SELECT ...]
[...]

-- Resep 7-4-3: Sintaks EXCEPT (kadang disebut juga MINUS)
--   Catatan: belum didukung oleh MySQL
SELECT ...
 EXCEPT [ALL] SELECT ...
[EXCEPT [ALL] SELECT ...]
[...]
```

Perintah-perintah SELECT yang ingin digabungkan tidak boleh mengandung klausa LIMIT atau ORDER BY. Dan tentu saja strukturnya harus sama (jumlah kolom, nama dan tipenya). UNION adalah operasi gabungan, dengan tambahan akan menghapus semua duplikat yang mungkin terbentuk. UNION ALL sama seperti UNION tapi tanpa menghapus duplikat. INTERSECT adalah operasi irisan, yakni hanya mengambil baris-baris yang sama yang ada di antara kedua query. Sementara EXCEPT/MINUS adalah operasi pengurangan dan hanya akan menghasilkan baris-baris di query pertama yang tidak ada pada query kedua.

Semua ini lebih mudah dijelaskan menggunakan contoh. Catatan: untuk INTERSECT dan EXCEPT hasilnya "fiktif" karena saat ini sebetulnya belum didukung MySQL.

```
DROP TABLE IF EXISTS t1;
CREATE TABLE t1 (i INT);
INSERT INTO t1 VALUES (1);
INSERT INTO t1 VALUES (1);
```

```
INSERT INTO t1 VALUES (2);
INSERT INTO t1 VALUES (3);
INSERT INTO t1 VALUES (3);

DROP TABLE IF EXISTS t2;
CREATE TABLE t2 (i INT);
INSERT INTO t2 VALUES (3);
INSERT INTO t2 VALUES (3);
INSERT INTO t2 VALUES (4);
INSERT INTO t2 VALUES (4);
INSERT INTO t2 VALUES (5);

mysql> SELECT * FROM t1 UNION SELECT * FROM t2;
+------+
| i    |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|    5 |
+------+
5 rows in set (0.00 sec)

mysql> SELECT * FROM t1 UNION ALL SELECT * FROM t2;
+------+
| i    |
+------+
|    1 |
|    1 |
|    2 |
|    3 |
|    3 |
|    3 |
|    3 |
|    4 |
|    4 |
|    5 |
+------+
10 rows in set (0.00 sec)
```

```
steven=# SELECT * FROM t1 INTERSECT SELECT * FROM t2;
+------+
| i    |
+------+
|    3 |
+------+
1 row in set (0.00 sec)

mysql> SELECT * FROM t1 INTERSECT ALL SELECT * FROM t2;
+------+
| i    |
+------+
|    3 |
|    3 |
+------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM t1 EXCEPT SELECT * FROM t2;
+------+
| i    |
+------+
|    1 |
|    2 |
+------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM t1 EXCEPT ALL SELECT * FROM t2;
+------+
| i    |
+------+
|    1 |
|    1 |
|    2 |
+------+
3 rows in set (0.00 sec)
```

Pada umumnya UNION/INTERSECT/EXCEPT lebih jarang dipakai ketimbang JOIN dan subselect, kecuali kalau Anda memisah-misah tabel yang sama ke dalam beberapa tabel fisik. Misalnya, untuk tabel jurnal transaksi Anda menulis ke tabel baru setiap bulannya (*transaksi_200409* untuk bulan Sep 2004, *transaksi_200410* untuk bulan Okt 2004, dan seterusnya) lalu Anda diminta membuat laporan untuk

transaksi beberapa bulan sekaligus. Desain seperti ini sebetulnya kurang dianjurkan menurut model relasional, dan ada cara-cara lain yang bisa dilakukan untuk mempercepat query ke tabel *transaksi* yang amat besar (jutaan baris atau lebih) seperti table partitioning di Oracle atau engine MERGE di MySQL. Namun cakupan mengenai hal ini di luar pembahasan buku.

7.5 Latihan

Pada latihan Bab 7 ini kita akan memfokuskan pada JOIN dan subselect. Kita akan menggunakan database yang kita buat di Bab 4 lalu, yaitu database nilai siswa. Skema database siswa ini sudah cukup ternormalisasi, yakni data-data sudah terpisah ke dalam beberapa tabel sebagaimana seharusnya. Untuk mengingatkan kembali pada struktur database siswa, saya persilakan Anda membaca-baca dulu jawaban latihan Bab 4. Keseluruhannya terdapat 11 tabel pada database.

Soal

Asumsi: Tahun ini = 2004 (tahun lalu = 2003).

1. Untuk kelas 6, siapa saja nama pengajar pelajarannya tahun ini?
2. Guru mana yang tidak menjadi wali kelas tahun ini?
3. Guru mana saja yang mengajar mata pelajaran yang berbeda tahun ini dibandingkan dengan tahun lalu? Catatan: seandainya seorang guru mengajar Matematika dan IPA tahun lalu, dan mengajar Matematika dan Bahasa Inggris tahun ini, maka guru tersebut termasuk. Namun jika seorang guru hanya mengajar pada salah satu tahun 2003 atau 2004 (tidak di keduanya), maka ini tidak termasuk.
4. Murid kelas 6 mana saja yang nilai rapor caturwulan 1 tahun 2004 di bawah 6,0? Sebutkan id dan nama mereka serta nilai rapor masing-masing.

Jawaban Latihan

Jawaban pertanyaan #1: data yang diminta ada pada tabel *guru_pelajaran*. Namun nama guru dan nama pelajaran tidak ada pada tabel ini melainkan masing-masing ada di tabel *guru* dan *pelajaran*. Karena itu kita melakukan dua buah LEFT JOIN sbb:

```
SELECT
  pelajaran.nama,
  guru.nama
FROM guru_pelajaran
  LEFT JOIN guru ON guru_pelajaran.id_guru = guru.id
  LEFT JOIN pelajaran ON guru_pelajaran.kodepelajaran =
pelajaran.kode
```

```
WHERE
  tingkat = 6 AND
  tahun = 2004;
```

Jawaban pertanyaan #2: data yang diminta ada pada tabel *walikelas* (atau tepatnya *harus tidak ada* pada tabel *walikelas*). Kita harus mencari guru yang tidak ada pada tabel tersebut. Ini dapat dilakukan dengan subquery kolumnar sebagai berikut:

```
SELECT
  nama
FROM guru
WHERE id NOT IN (SELECT id_guru FROM walikelas WHERE tahun=2004);
```

Jawaban pertanyaan #3: Salah satu cara menyelesaikan permasalah ini adalah dengan membuat dulu daftar pelajaran yang diajar oleh setiap guru untuk tahun 2003:

```
SELECT
  id_guru,
  GROUP_CONCAT(DISTINCT kodepelajaran
               ORDER BY kodepelajaran) AS pelajaran
FROM guru_pelajaran WHERE tahun=2003 GROUP BY id_guru;
```

Jika ada seorang guru (mis: id=1) mengajar dua mata pelajaran, mis: MAT (Matematika) dan IPA maka hasilnya adalah: (1, 'IPA-MAT').

Kita lalu membuat query yang sama untuk tahun 2004:

```
SELECT
  id_guru,
  GROUP_CONCAT(DISTINCT kodepelajaran
               ORDER BY kodepelajaran) AS pelajaran
FROM guru_pelajaran WHERE tahun=2004 GROUP BY id_guru;
```

Selanjutnya kita dapat menggabungkan kedua hasil ini menggunakan INNER JOIN USING (id_guru) untuk membandingkan pelajaran tahun 2003 dan 2004 untuk setiap guru. Kita lalu dapat menambahkan kondisi `kiri.pelajaran <> kanan.pelajaran` untuk mencari yang mengajar pelajaran yang berbeda. Terakhir kita tinggal melakukan JOIN dengan tabel *guru* untuk mengambil nama guru. Query akhir berbentuk:

```
SELECT
  guru.nama,
  a2003.pelajaran AS p2003, a2004.pelajaran AS p2004
FROM (
  SELECT
    id_guru,
    GROUP_CONCAT(DISTINCT kodepelajaran
                 ORDER BY kodepelajaran) AS pelajaran
  FROM guru_pelajaran WHERE tahun=2003 GROUP BY id_guru
) AS a2003
JOIN (
  SELECT
    id_guru,
    GROUP_CONCAT(DISTINCT kodepelajaran
                 ORDER BY kodepelajaran) AS pelajaran
  FROM guru_pelajaran WHERE tahun=2004 GROUP BY id_guru
) AS a2004
USING(id_guru)
LEFT JOIN
  guru ON a2003.id_guru = guru.id
HAVING p2003 <> p2004;
```

Dalam menyusun query yang agak kompleks seperti di atas, caranya adalah sepotong demi sepotong. Pertama kita mengetes masing-masing tabel subselect a2003 dan a2004 yang akan kita JOIN. Lalu kita mengecek hasil JOIN-nya tanpa klausa HAVING. Baru terakhir kita melakukan JOIN dengan tabel *guru*.

Jawaban pertanyaan #4: Untuk mencari jawaban ini pertama kita menghitung nilai rapor murid untuk caturwulan 1 tahun 2004 yang nilai rata-ratanya di bawah 6,0:

```
SELECT
  id_siswa,
  AVG(nilai) AS nilai_rapor
FROM nilai
WHERE tahun=2004 AND caturwulan=1
  -- sebetulnya 'GROUP BY id_siswa' saja juga bisa, karena kita
hanya
  --   memerlukan 1 nilai tunggal untuk tahun dan caturwulan, dan
  --   ini sudah difilter di klausa WHERE
GROUP BY id_siswa, tahun, caturwulan
HAVING nilai_rapor < 6.0;
```

Namun kita masih harus memfilter hanya mencari siswa kelas 6. Data ini ada pada tabel *ruangkelas_siswa* yang mencatat siswa mana ada di kelas mana dan *ruangkelas* yang mencatat ruang kelas mana merupakan ruang untuk tingkat berapa. Karena itu kita melakukan JOIN dengan *ruangkelas_siswa*:

```
SELECT
  nilai.id_siswa,
  AVG(nilai) AS nilai_rapor
FROM nilai
JOIN ruangkelas_siswa rs
  ON nilai.id_siswa = rs.id_siswa AND nilai.tahun = 2004
JOIN ruangkelas r
  ON rs.ruangkelas = r.nama AND tingkat = 6
WHERE nilai.tahun = 2004 AND caturwulan = 1
GROUP BY nilai.id_siswa
HAVING nilai_rapor < 6.0;
```

Kini yang kurang hanyalah nama si siswa, dan ini bisa diperoleh di tabel *siswa*, jadi kita tinggal melakukan JOIN dengan tabel *siswa*:

```
SELECT
  siswa.nama,
  nilai.id_siswa,
  AVG(nilai) AS nilai_rapor
FROM nilai
JOIN ruangkelas_siswa rs
  ON nilai.id_siswa = rs.id_siswa AND nilai.tahun = 2004
JOIN ruangkelas r
  ON rs.ruangkelas = r.nama AND tingkat = 6
JOIN siswa ON nilai.id_siswa = siswa.id
WHERE nilai.tahun = 2004 AND caturwulan = 1
GROUP BY nilai.id_siswa
HAVING nilai_rapor < 6.0;
```

Lagi-lagi, dalam menyusun query yang melibatkan banyak tabel seperti ini kita melakukannya langkah demi langkah. Lakukan JOIN satu persatu dan cek hasilnya.